<http://blog.csdn.net/Spwper/article/details/51351793>

版权声明：本文为博主原创文章，未经博主允许不得转载。

我们来看一个信号与槽的小例子   
头文件：

#ifndef MYCLASS

#define MYCLASS

#include <QObject>

class myClass : public QObject

{

Q\_OBJECT

public:

explicit myClass(QObject \*parent = 0 );

~myClass();

void triggerOne();

void triggerTwo();

signals:

void signalOne(QString); //自定义信号one

void signalTwo(int); //自定义信号two

private slots: //私有槽函数只能与自己关联

protected slots: //在共有保护派生子类可连接

public slots: //共有继承派生类可连接

QString slotOne(QString msg);

int slotTwo(int a);

};

#endif // MYCLASS

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23

实现文件：

#include <QApplication>

#include <QObject>

#include "myclass.h"

#include <iostream>

using std::cout;

using std::endl;

myClass::myClass(QObject \*parent):

QObject(parent)

{

//Qt4里面信号与槽的连接方式

// connect(this , SIGNAL(signalOne(QString)) , this ,SLOT(slotOne(QString) , Qt::AutoConnection);

// connect(this , SIGNAL(signalTwo(int)) , this ,SLOT(slotTwo(int) , Qt::AutoConnection);

//Qt5的连接方式，推荐使用。大家可以去看下这两者的区别

connect(this , &myClass::signalOne , this , &myClass::slotOne , Qt::AutoConnection);

connect(this , &myClass::signalTwo , this , &myClass::slotTwo , Qt::AutoConnection);

triggerOne();

triggerTwo();

}

myClass::~myClass()

{

}

void myClass::triggerOne()

{

emit signalOne("SignalOne");

}

void myClass::triggerTwo()

{

emit signalTwo(88);

}

QString myClass::slotOne(QString msg)

{

cout<<"this is slot one::"+msg.toStdString()<<endl;

return msg;

}

int myClass::slotTwo(int a)

{

cout<<"this is slot two::"<<a<<endl;

return a;

}

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

myClass test;

return a.exec();

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51

在写这个小例子过程中，小白脑残的本来想写在一个cpp文件中。最后发现不行。查资料才知道，moc工具识别Q\_OBJECT宏定义是在头文件中去识别的，些就是说在cpp文件里面包含这个宏是没有用的。详细看此贴<http://blog.chinaunix.net/uid-20801802-id-1839159.html>   
帮助里面的“read a C++ source file”也只是形象的说法。

下面我们来慢慢分析Q\_OBJEC是怎么实现moc\_myclass.cpp文件的。   
1、我们看一下Q\_OBJECT在Qt库文件（5.5版本）的定义。

#define Q\_OBJECT \

public: \

Q\_OBJECT\_CHECK \

QT\_WARNING\_PUSH \

Q\_OBJECT\_NO\_OVERRIDE\_WARNING \

static const QMetaObject staticMetaObject; \

virtual const QMetaObject \*metaObject() const; \

virtual void \*qt\_metacast(const char \*); \

virtual int qt\_metacall(QMetaObject::Call, int, void \*\*); \

QT\_WARNING\_POP \

QT\_TR\_FUNCTIONS \

private: \

Q\_DECL\_HIDDEN\_STATIC\_METACALL static void qt\_static\_metacall(QObject \*, QMetaObject::Call, int, void \*\*); \

struct QPrivateSignal {};

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

moc的第一工作肯定是统计分析开发者写的这个类有哪些信号槽以及各自的参数情况：   
这里是moc生成moc\_mycalss.cpp文件信号槽数量数据代码：

struct qt\_meta\_stringdata\_myClass\_t {

QByteArrayData data[8];

char stringdata0[51];

};

#define QT\_MOC\_LITERAL(idx, ofs, len) \

Q\_STATIC\_BYTE\_ARRAY\_DATA\_HEADER\_INITIALIZER\_WITH\_OFFSET(len, \

qptrdiff(offsetof(qt\_meta\_stringdata\_myClass\_t, stringdata0) + ofs \

- idx \* sizeof(QByteArrayData)) \

)

static const qt\_meta\_stringdata\_myClass\_t qt\_meta\_stringdata\_myClass = {

{

QT\_MOC\_LITERAL(0, 0, 7), // "myClass"

QT\_MOC\_LITERAL(1, 8, 9), // "signalOne"

QT\_MOC\_LITERAL(2, 18, 0), // ""

QT\_MOC\_LITERAL(3, 19, 9), // "signalTwo"

QT\_MOC\_LITERAL(4, 29, 7), // "slotOne"

QT\_MOC\_LITERAL(5, 37, 3), // "msg"

QT\_MOC\_LITERAL(6, 41, 7), // "slotTwo"

QT\_MOC\_LITERAL(7, 49, 1) // "a"

},

"myClass\0signalOne\0\0signalTwo\0slotOne\0"

"msg\0slotTwo\0a"

};

#undef QT\_MOC\_LITERAL

static const uint qt\_meta\_data\_myClass[] = {

// content:

7, // revision

0, // classname

0, 0, // classinfocount，classinfodata

4, 14, // methodscount,methoddata

0, 0, // propertiescount,propertiesdata

0, 0, // enums/sets

0, 0, // constructors

0, // flags

2, // signalCount

// signals: name, argc, parameters, tag, flags

1, 1, 34, 2, 0x06 /\* Public \*/,

3, 1, 37, 2, 0x06 /\* Public \*/,

// slots: name, argc, parameters, tag, flags

4, 1, 40, 2, 0x0a /\* Public \*/,

6, 1, 43, 2, 0x0a /\* Public \*/,

// signals: parameters

QMetaType::Void, QMetaType::QString, 2,

QMetaType::Void, QMetaType::Int, 2,

// slots: parameters

QMetaType::QString, QMetaType::QString, 5,

QMetaType::Int, QMetaType::Int, 7,

0 // eod

};

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58

这里前面的数字注释都说的很清楚，基本数据的统计。在signals和slots里面的：1、3、4、6表示自己在字符串:”myClass\0signalOne\0\0signalTwo\0slotOne\0”   
“msg\0slotTwo\0a”中从第几个“\0”开始表示自己的信息。这里从偏移量开始signal的第一个”\0”后面表示信号名称，第二个”\0”表示返回值（同学要问了，信号函数实现都没有哪来的返回值，对于异步调用返回值永远是默认构造函数出来的，因为调用的函数都没执行完，只能默认一个返回值，对于同步调用，则是最后连接的那个函数的执行的返回值，如果中间有不匹配的情况，Qt会帮你解决，默认了），同理后面的slots更好理解了。   
（1）**static const QMetaObject staticMetaObject;**   
其对应的moc文件部分（set）：

const QMetaObject myClass::staticMetaObject = {

{ &QObject::staticMetaObject, qt\_meta\_stringdata\_myClass.data,

qt\_meta\_data\_myClass, qt\_static\_metacall, Q\_NULLPTR, Q\_NULLPTR}

};

//staticMetaObject静态值是由Q\_OBJECT引出的，这里只对其自定义部分的参数进行赋值保存，QMetaObject结构体远比这个复杂，只是其他部分与开发者自定义内容无关，moc编译器会自行处理，所以这里生成的moc\_myclass.cpp不做处理：

struct { // private data

const QMetaObject \*superdata; //父类信息

const QByteArrayData \*stringdata;

const uint \*data;

typedef void (\*StaticMetacallFunction)(QObject \*, QMetaObject::Call, int, void \*\*);

StaticMetacallFunction static\_metacall;

const QMetaObject \* const \*relatedMetaObjects;//其他关联类信息

void \*extradata; //reserved for future use

};

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

看别人的博客是没有：   
typedef void (StaticMetacallFunction)(QObject , QMetaObject::Call, int, void \*\*);   
StaticMetacallFunction static\_metacall;   
const QMetaObject \* const \*relatedMetaObjects;   
这三项的，不知道别人使用的是哪个版本。讲道理的话这里，在多类继承的情况下，是有记录关联类信息的，但看代码也只有一个。但是在多类继承的时候，想要拥有QObject的属性时候，具有QObject属性的父类必须放在第一位。大牛说moc规定多继承的情况下，moc会假设第一个继承的类为QObject, 并且必须要保证在多继承中，只有唯一一个类是继承自QObject的。这样看上去，多余一个QObject继承的，第二个QObject根本没办法识别出来。这里小白只知道要放在第一个，不清楚moc的具体工作流程。   
get部分：

const QMetaObject \*myClass::metaObject() const

{

return QObject::d\_ptr->metaObject ? QObject::d\_ptr->dynamicMetaObject() : &staticMetaObject;

}

* 1
* 2
* 3
* 4

这里的d\_ptr是一个Qt里面的模板类指针，并不是QObject的静态成员，这里我在帮助文档里也没找到dynamicMetaObject()成员函数，猜测是一个安全处理，返回staticMetaObject。   
**（2）virtual void qt\_metacast(const char );**

void \*myClass::qt\_metacast(const char \*\_clname)

{

if (!\_clname) return Q\_NULLPTR;

if (!strcmp(\_clname, qt\_meta\_stringdata\_myClass.stringdata0))

return static\_cast<void\*>(const\_cast< myClass\*>(this));

return QObject::qt\_metacast(\_clname);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

当传入的字符串数据是当前这个类的话，就将this指针转换成void指针传给外界： 这个操作相当危险。

如果不是当前类的话，就调用父类的qt\_metacast继续查询。

在这里，我的父类是QObject，所以自然就调用QObject::qt\_metacase了。   
**（3）virtual int qt\_metacall(QMetaObject::Call, int, void**);\*\*

void myClass::qt\_static\_metacall(QObject \*\_o, QMetaObject::Call \_c, int \_id, void \*\*\_a)

{

if (\_c == QMetaObject::InvokeMetaMethod) {

myClass \*\_t = static\_cast<myClass \*>(\_o);

Q\_UNUSED(\_t)

switch (\_id) {

case 0: \_t->signalOne((\*reinterpret\_cast< QString(\*)>(\_a[1]))); break;

case 1: \_t->signalTwo((\*reinterpret\_cast< int(\*)>(\_a[1]))); break;

case 2: { QString \_r = \_t->slotOne((\*reinterpret\_cast< QString(\*)>(\_a[1])));

if (\_a[0]) \*reinterpret\_cast< QString\*>(\_a[0]) = \_r; } break;

case 3: { int \_r = \_t->slotTwo((\*reinterpret\_cast< int(\*)>(\_a[1])));

if (\_a[0]) \*reinterpret\_cast< int\*>(\_a[0]) = \_r; } break;

default: ;

}

} else if (\_c == QMetaObject::IndexOfMethod) {

int \*result = reinterpret\_cast<int \*>(\_a[0]);

void \*\*func = reinterpret\_cast<void \*\*>(\_a[1]);

{

typedef void (myClass::\*\_t)(QString );

if (\*reinterpret\_cast<\_t \*>(func) == static\_cast<\_t>(&myClass::signalOne)) {

\*result = 0;

}

}

{

typedef void (myClass::\*\_t)(int );

if (\*reinterpret\_cast<\_t \*>(func) == static\_cast<\_t>(&myClass::signalTwo)) {

\*result = 1;

}

}

}

}

看名字也知道这里该类所有信号处理函数，并不是我们常用的slots。当我们emit一个信号时，也是调用该函数再调用真正的信号函数，当其他关联信号过来是，才是调用真正的槽函数。

真正的信号函数：

// SIGNAL 0

void myClass::signalOne(QString \_t1)

{

void \*\_a[] = { Q\_NULLPTR, const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t1)) };

QMetaObject::activate(this, &staticMetaObject, 0, \_a);

}

// SIGNAL 1

void myClass::signalTwo(int \_t1)

{

void \*\_a[] = { Q\_NULLPTR, const\_cast<void\*>(reinterpret\_cast<const void\*>(&\_t1)) };

QMetaObject::activate(this, &staticMetaObject, 1, \_a);

}

我在帮助里并没有找到activate函数，看大牛们的说的就是：当执行流程进入activate中，会先从connectionLists中取出这个signal相对应的connection list，而这个总的连接表是由connect（）函数完成的，下章我们会讲到。看上面两个函数的第三个参数，代表的就是这个类的第几个信号的索引，类的信号槽索引是一起编排的。

很多地方小白也是半知半解，不知道讲清楚没有。到这里moc就是把Qt的信号与槽的那部分相关代码转化为C++代码。如果有大牛，求教育啊。。